OIPE JC174 AUG 23 2001 PATENT & TRADEMARK OFFICE

2631 RS #2

10-15-01

RECEIVED
AUG 24 2001
Technology Center 2600

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of :	DALVI, Aneesh, et al.
Serial No.:	09/811,566
Group:	Unknown
Confirmation No.:	Unknown
Filed:	May 2, 2001
Examiner:	Unknown
Title:	METHOD AND SYSTEM FOR PROVIDING ENGINEERING ANALYSIS TOOLS IN A DISTRIBUTED ENVIRONMENT
Attorney Ref:	PAT 240-2

The Honourable Commissioner of Patents
and Trademarks
Washington, D.C. 20231
U.S.A.

Dear Sirs:

Please find enclosed herewith for submission the certified copy of Canadian Patent Application No. 2,301,440 on which United States Patent Application No. 09/811,566 claims priority.

Respectfully submitted,
DALVI, Aneesh, et al.

August 21, 2001
Date

By: Anne Kinsman
Registration No. 45291
BORDEN LADNER GERVAIS LLP
60 Queen Street
Ottawa, ON K1P 5Y7
Telephone 613-237-5160
Facsimile 613-787-3558
E-mail ipott@blgcanada.com

ALK/cdg

Encl.
1. Certified Copy CA Appln. No. 2,301,440

Office de la propriété intellectuelle du Canada
Un organisme d'Industrie Canada

Canadian Intellectual Property Office
An Agency of Industry Canada

RECEIVED
AUG 24 2001
Technology Center 2600

OIPE
AUG 23 2001
PATENT & TRADEMARK OFFICE

Bureau canadien des brevets

Certification

Canadian Patent Office

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,301,440**, on March 20, 2000, by **SPACEBRIDGE NETWORKS CORPORATION**, assignee of Aneesh Dalvi and Amal Khailtash, for "Method and System for Configuring an Air Interface in a Modem".

S. Gregoire

Agent certificateur/Certifying Officer

June 26, 2001

Date

Canada

(CIPO 68)
01-12-00

OPIC CIPO

ABSTRACT

A system and method for configuring an air interface processor in a modem. The air interface processor includes an event handler and a microsequencer. The event handler schedules data for transmission and reception in the modem, and the microsequencer sends commands to a frame formatter based on the schedule.

METHOD AND SYSTEM FOR CONFIGURING AN AIR INTERFACE IN A MODEM

FIELD OF THE INVENTION

The present invention relates to a method and system for data transmission in a modem. In particular, the present invention relates to a method and system for configuring an air interface for data transmission in a modem according to multiple standards.

BACKGROUND OF THE INVENTION

Generally, modems have been designed to work with one of many existing standards. For example, while there are many air interface, or radio link, standards, most current modems are designed to operate with only one of them. Even within a particular standard, uplink and downlink formats are very different, and require separately designed air interface processors. Clearly, this results in increased costs for both design and hardware when a new standard is implemented, and precludes using a modem designed to work with one standard from being reconfigured to work with a different standard.

It is therefore desirable to provide an air interface processor that can be configured for more than one standard, or data format.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a method and system that obviates or mitigates at least one disadvantage of the prior art. It is a particular object of the present invention to provide a method and system for configuring an air interface in a modem for multiple data and transmission standards and formats.

In a first aspect, the present invention provides an air interface processor for a modem, comprising:

an event handler for scheduling the processing of data transmission and reception in the modem;

a microsequencer receiving instructions from the event handler and determining commands to send to a frame formatter in the modem.

According to a further aspect of the present invention, there is provided a method for processing data, comprising the steps of:

(i) scheduling the processing of data transmission and reception;

(ii) transmitting the schedule to a microsequencer;

(iii) sending commands to a frame formatter to build a frame of data in accordance with the schedule.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, with reference to the attached Figures, wherein:

Figure 1 is a block diagram of a modem incorporating an air interface processor according to the present invention;

Figure 2 is a block diagram of an air interface processor according to the present invention;

Figure 3 is a chart of event handler instruction sets according to the present invention;

Figure 4 is a chart of general purpose instructions according to the present invention;

Figure 5 is a chart of event handler ALU opcodes according to the present invention;

Figure 6 is a chart of event handler branch conditions according to the present invention;

Figure 7 is a chart of register access instructions according to the present invention;

Figure 8 is a chart of data scheduling instructions according to the present invention;

Figure 9 is a chart of burst descriptor instructions according to the present invention;

Figure 10 is a chart of a modulator burst information field format according to the present invention;

Figure 11 is a chart of a demodulator burst information field format according to

the present invention;

Figure 12 is a chart of a processor wait instruction according to the present invention;

Figure 13 is a chart of a microsequencer instruction set according to the present invention;

Figure 14 is a chart of a microsequencer memory format according to the present invention;

Figure 15 is a block diagram of a configuration of condition codes and forks according to the present invention; and

Figure 16 is an example of fork values according to the present invention;

DETAILED DESCRIPTION OF THE INVENTION

A modem incorporating an air interface processors (AIPs) in the modulator and demodulator, respectively, is shown in Fig. 1. Both the Modulator and the Demodulator have special-purpose processors that are designed with the intention of allowing the Modem to meet any air interface standard. However, it is impossible to predict whether all forthcoming LMDS and SatCom systems can be handled. For any systems which cannot be handled directly, there will be a provision to bypass internal FEC generation/correction and process a raw bit stream.

The Air Interface Processor, is generally shown in Fig. 2, and is divided into a transmit-side and receive-side unit. Each Processor Unit consists of an *Event Handler* and a *Microsequencer*. The Event Handler provides an abstraction of the Burst Frequency Time Plan; the Microsequencer controls how data is formatted by the Modulator/Demodulator circuitry.

The configuration of the Microsequencer is intended to be static during the course of operation (though this is not a necessary condition). The Event Handler configuration is static for continuous mode operation, and dynamic for burst mode to accommodate varying Burst-Frequency Time Plans.

The primary purpose of the Event Handler is to schedule the processing of burst data transmission/arrival in the Modem. (Continuous data is treated as a special subset of burst data.) In addition to being able to initiate sending/receiving bursts of data, the Event Handler has the ability to perform various ALU operations, and to perform conditional or unconditional branches. The ALU and branch operations are performed simultaneously, which allows for

greater code density than would otherwise be possible.

The instruction set is divided into *general purpose* and *modem control* instructions. There are 8 16-bit read/write registers, and 8 16-bit read-only registers. All instructions occupy one 48-bit word in memory. The instruction set summary is shown in Fig. 3. The Event Handler memory is a single-port, synchronous, 1K * 48 RAM.

General-purpose instructions (bits 47:46=00), as shown in Fig. 4, are comprised of an ALU instruction and a two-way branch instruction. The ALU instructions are reminiscent of the ARM RISC processor, in that the second operand always passes through a programmable barrel shifter before being applied to the ALU.

For each operation, the ALU computes a new result and the flags associated with that result (N=negative, Z=zero, C=carry, V=overflow). The results of the ALU operation are stored in the destination register only if the 'W' flag is asserted for that instruction. The ALU flags are updated only if the 'S' flag is asserted.

The results of the ALU operation are written to a destination register (Rd). The destination register may be any of the 8 read/write registers (R0-R7). Operand 1 (Rn) is always a register, and may be any of the 16 registers. Operand 2 is either an immediate value (I=1) or a register value (I=0), and is passed through a programmable barrel shifter to yield a 16-bit result.

In general, each ALU instruction performs the following operation:

```
result = fn(Rn, shift(op2))
   Rd        <= result if W else no change
  {N,Z,C,V} <= flags(result) if S else no change
```

The operations supported are shown in Fig. 5.

The branch component of the instruction allows a conditional branch to happen based on the result of the *previous* ALU instruction. Each branch instruction has an address offset to jump to in the case the condition passes, and a separate address offset to jump to in the case the condition fails. This mechanism only allows for relative branches.

The branch codes supported are shown in Fig. 6.

Register Access instructions (bits 47:46=01), as shown in Fig. 7, are provided to allow reading or writing arbitrary registers within the Modem. The Modulator Air Interface Processor can only control registers within the Modulator, and the Demodulator Air Interface Processor can only control registers within the Demodulator. This facility is used to

enable/disable internal ASIC blocks, and to control external devices that need to be manipulated on a real-time basis, such as RF frequency select for MF-TDMA systems. It is not intended for static configuration, which is better handled via the processor interface (although it can be performed here as well).

Rhi provides the top 16 bits of the 24 bit address. Rlo (bit 44=0) or imm_lo (bit 44=1) provides the bottom 8 bits. For a write operation (bit 45=0), the 32 bit data is contained directly in the lower bits of the instruction. For a read operation (bit 45=1), the bottom 3 bits encode the number of the destination register. The register access occurs over the internal HCPU bus.

Data Scheduling instructions (bit 47=1), as shown in Fig. 8, are the means by which bursts of data are transmitted or received by the Modem. Each of these instructions is an entry that maps time indices to actions. Time indices are specified in clock ticks relative to the start of a super-frame. Time index 0 is determined via the PCR algorithm (PCR offset). Actions are pointers to microcode instruction sequences. With this scheme, it is not necessary to count frames or even timeslots, only superframes.

For the execute phase of a data scheduling instruction, the Event Handler will wait until the current time (time index relative to super-frame start) is approximately equal to the trigger time. (It can only wait for the times to be approximately equal because the Event Handler runs off the byte clock, whereas the current time is a counter that runs off the sample clock.) When the trigger time is reached, a start command is sent to the Microsequencer, which will begin running at the address specified in "microsequencer address".

A time offset of 32'hFFFF_FFFF is a special code indicating that this event is to be processed immediately. The microsequencer address 0 is a special code indicating that the start command should not be sent.

the trigger flag is set, the time offset in the current event is passed to the Preamble Insert module in the Modulator, or Direct Sampling module in the Demodulator.
Bursts can be conditional on the availability of data in a certain queue. In this case, the 'D' flag must be set to 1, and 'Q' must be set to the number of the data queue which must contain data (0 → DATA1, 1 → DATA2, etc).

For burst mode applications, the AIP provides a special BURST instruction, as shown in Fig. 9, which is used to specify special information related to the following burst of

data.

The contents of the burst info field are different for the modulator and demodulator. For the modulator, as shown in Fig. 10, this field is used to select which preamble is to be used for the following burst (PS). Up to four preambles may be defined. For MF-TDMA applications, the burst info field contains frequency information that is used to reprogram the DDS or Fractional-N counter.

For the demodulator, as shown in Fig. 11, the burst info field is used to select the expected preamble for the incoming burst (PS). Up to four preambles may be defined. It also contains the length of the expected burst. The SB7016 tags all incoming bursts with an arbitrary user ID, which the MAC layer software can use to correlate received bursts with expected bursts in the BFTP. The user ID is specified in the burst info field.

Fig. 12 shows a typical processor wait instruction.

The Microsequencer is responsible for sending commands to the Frame Formatter. On the Modulator, this builds up a frame of data on a byte-by-byte basis.

The core of the Microsequencer design is based on a modified version of the AMD2910 micro-program sequencer. The original 2910 is a 12 bit sequencer with a 32 word stack, and is capable of conditional branching, subroutine calls, and looping. The microsequencer in the Air Interface Processor is a 10 bit sequencer, but adds some powerful instructions, as shown in Fig. 13, to perform multi-way conditional branching, among other things.

The number of microcode sequences is limited only by the available Sequence RAM (SeqRAM) available. The SeqRAM is a single-port synchronous 1K * 32 RAM.

The Microcode Sequencer has a program counter (PC), which is initially set by the Event Handler. When it is instructed to start by the Event Handler, it shall begin executing the instructions in SeqRAM at the specified address, until such time as a JZ (jump-to-zero) instruction is found.

The memory format for the modulator microsequencer is shown in Fig. 14.

The EMIT field is used as an argument to the OPCODE field, and is used for branch addresses or to load the internal counter. The SR field (Scrambler reset) can be used to reset the Scrambler at any time.

The FORK instruction, which is an addition to basic 2910 instruction set, uses the

value formed by the fork_cc(3:0) vector as the basis for a 16-way branch. For example, if the value of fork_cc is 12, the microsequencer will advance its program counter by 12.

The fork_cc value is updated every clock cycle based on the configuration circuitry shown in Fig. 15. Each bit in fork_cc is derived from a programmable look-up table. A 5-bit value is used as the index to this look-up table.

FORK LUT 0	insert_pcr	end of message in queue 1	counter < threshold	counter == threshold	r0
FORK LUT 1	insert_pcr	end of message in queue 2	counter < threshold	counter == threshold	r1
FORK LUT 2	insert_pcr	end of message in queue 3	counter < threshold	counter == threshold	r2
FORK LUT 3	insert_pcr	end of message in queue 4	counter < threshold	counter == threshold	r3

Table 1: FORK LUT Indices

Programming the FORK instruction is a rather convoluted process, as it involves three layers of indirection. The FORK instruction is essentially a "case" statement. The idea is to generate a 4-bit value (fork_cc) to create an offset of 0 to 15. Each bit of fork_cc is derived from the corresponding FORK LUT. Each FORK LUT is indexed by the 5-bit number formed by the concatenation of the conditions shown in Table 1. The conditions r0, r1, r2, and r3 are bits from the R7 register in the Event Handler. The R7 Bit Select register can be used to select among the lower 8 bits of the R7 register. Refer to Figure 15 for more details.

For example, suppose we wanted fork_cc(0) to evaluate as true whenever insert_pcr and counter==threshold are both true. This condition can be expressed as 1xx1x, which means that bits 4 and 1 must be 1, and the other bits are don't cares. In this case, we would program the bits in the LUT that match 1xx1x with 1, as shown in Fig. 16, and the others with 0. The value generated is 32'hCCCC_0000.

The condition codes (CCSEL) for conditional branches (for example, CJV) in the Microsequencer are hard-coded as follows:

Condition Code	Description
0	always false
1	counter != 0
2	counter == threshold
3	counter < threshold
4	counter <= threshold
5	insert_pcr == 1
6	end of message in queue 1
7	end of message in queue 2
8	end of message in queue 3
9	end of message in queue 4
10	insert_pcr ==1 AND end of message in queue 1
11	insert_pcr ==1 AND end of message in queue 2
12	insert_pcr ==1 AND end of message in queue 3
13	insert_pcr ==1 AND end of message in queue 4
14	value of R7(bit 9)
15	value of R7(bit 8)

Register R7 in the Modulator Event Handler can be used to control the Microsequencer to some degree. Bits 8 and 9 of the R7 register are used to generate condition codes 14 and 15. Bits 0 to 7 of the R7 register, in conjunction with the static R7 Bit Select register, can be used as inputs to the FORK LUT.

The read-only Event Handler registers for the Modulator contain the following values:

R8	Bytes remaining in message, queue 1
R9	Bytes remaining in message, queue 2
R10	Bytes remaining in message, queue 3
R11	Bytes remaining in message, queue 4
R12	Condition Code Register
R13	Event Handler Program Counter
R14	32'h0000_0000
R15	32'hFFFF_FFFF

The definition of FFCMD (Frame Formatter command) is shown in Table 2. This instruction set is used to generate frames to be fed into the modulator Frame Formatter. As data or control words are inserted, they can be flagged with an attribute indicating whether or not they should be scrambled, and which outer code is to be used (ie. Reed-Solomon, CRC, or neither). If the SB bit is asserted in the Microsequencer instruction, the Scrambler is bypassed. The value in the OC field is used to select which outer coding scheme is to be used (0-3).

The Frame Formatter is also able to insert register values from the Event Handler into the data stream. This mechanism allows the generated data stream to be more dynamic than would otherwise be possible. When the Event Handler issues a start command to the Microsequencer, the contents of R0 to R7 are passed through a programmable barrel shifter and

stored as inputs to the Frame Formatter.

Command	Description	Name
0	nop	NOP
1 – 16	Insert Control Word 1 – 16	CW1 – CW16
17 – 24	Insert Shifted Register 0 – 7	R0 – R7
25	Insert Original PCR	PCRO
26	Insert Current PCR	PCR
27	Insert Data from Queue 1	DATA1
28	Insert Data from Queue 2	DATA2
29	Insert Data from Queue 3	DATA3
30	Insert Data from Queue 4	DATA4
31	Flush	FLUSH

Table 2: Modulator Command Set for Frame Formatter

The most common use of the barrel shifter is to align an 8-bit value in one of the registers to the upper 8 bits of the 16-bit register value (inputs to the Frame Formatter must be MSB-aligned). The barrel shifter is configured as follows:

R7 shift	R6 shift	R5 shift	R4 shift	R3 shift	R2 shift	R1 shift	R0 shift

Each Rx shift field is defined as follows:

2'b00 = LSL (Rn, shift_by) 2'b01 = LSR (Rn, shift_by) 2'b10 = ASR (Rn, shift_by) 2'b11 = ROL (Rn, shift_by)	shift_by

A simple microcode sequence to generate an MPEG frame for a base station could be programmed in the following manner. This sequence indicates that all bytes except for the initial sync should be scrambled (".S")

```
MPEG_FRAME:
        CONT            CW1.OC1         # 1 byte, typically 47h
        CONT            CW2.S.OC1               # 1 byte
        CONT            CW3.S.OC1               # 1 byte
        CONT            CW4.S.OC1               # 1 byte
        LDCT    182     DATA2.S.OC1             # send data from queue 2, load counter
loop:   RPCT    loop    DATA2.S.OC1             # continue pulling data from queue 2
                                        # until counter reaches zero; total = 184
        JZ              FLUSH           # indicates end of burst; soft reset
```

The device can act as a simple ATM segmentation engine by setting the control

word to a 5-byte ATM header. Up to 16 simultaneous ATM connections can be handled in this manner (one per Control Word).

```
ATM_CONN_1:
        CONT            CW1.S.OC1               # CW1 = 5-bytes = { VPI/VCI=(a,b) }
        LDCT    2       NOP             # 4 NOPs because CW1 is 5 bytes
loop1:  RPCT    loop1   NOP
        LDCT    46      DATA1.S.OC1             # send 48 bytes from queue 1
loop2:  RPCT    loop2   DATA1.S.OC1
           JZ           FLUSH
```

The above-described embodiments of the invention are intended to be examples of the present invention. Alterations, modifications and variations may be effected the particular embodiments by those of skill in the art, without departing from the scope of the invention which is defined solely by the claims appended hereto.

We claim:

1. An air interface processor for a modem, comprising:

an event handler for scheduling the processing of data transmission and reception in the modem;

a microsequencer receiving instructions from the event handler and determining commands to send to a frame formatter in the modem.

2. A method for processing data, comprising the steps of:

(i) scheduling the processing of data transmission and reception;

(ii) transmitting the schedule to a microsequencer;

(iii) sending commands to a frame formatter to build a frame of data in accordance with the schedule.

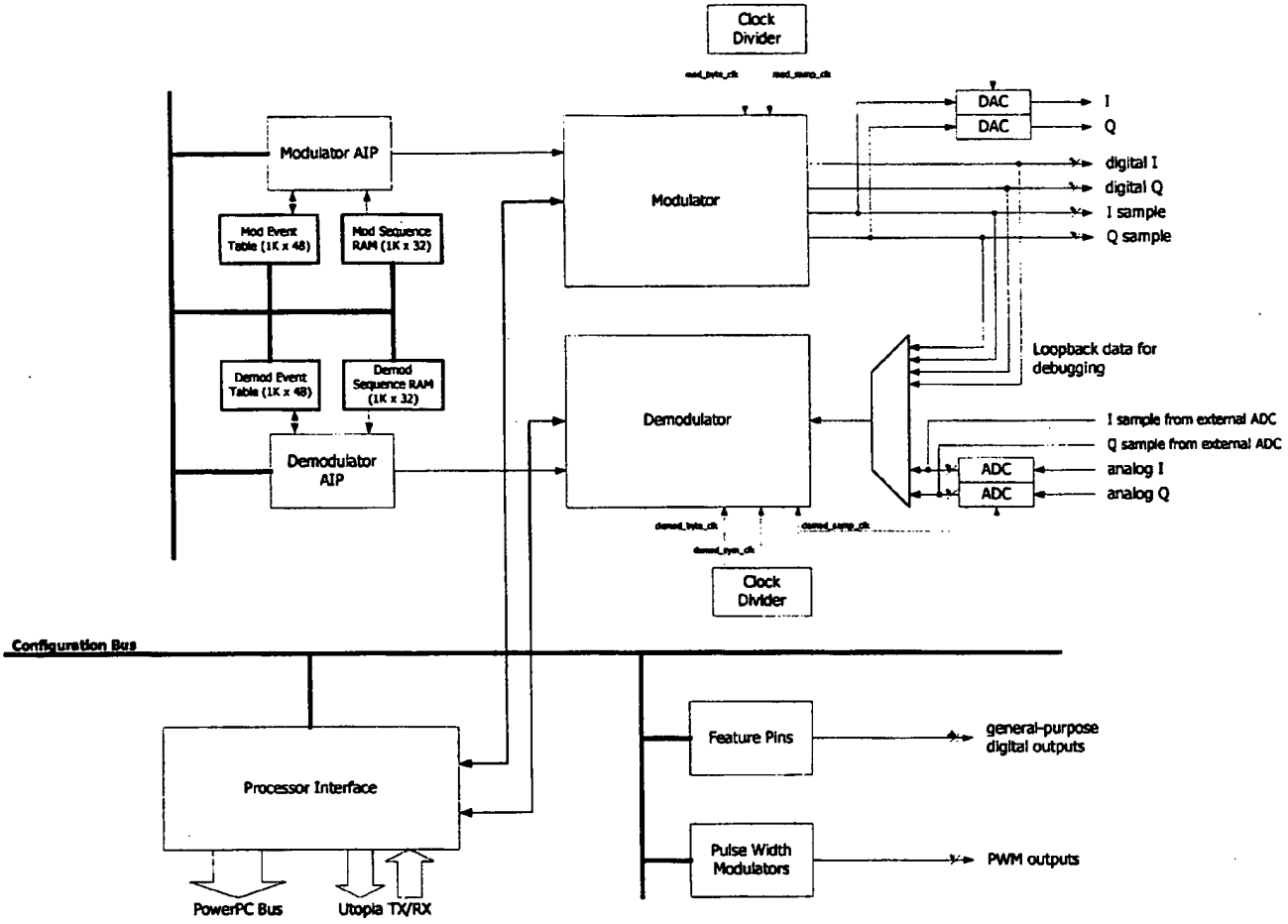

Figure 1

Borden Ladner Gervais LLP / s.r.l.

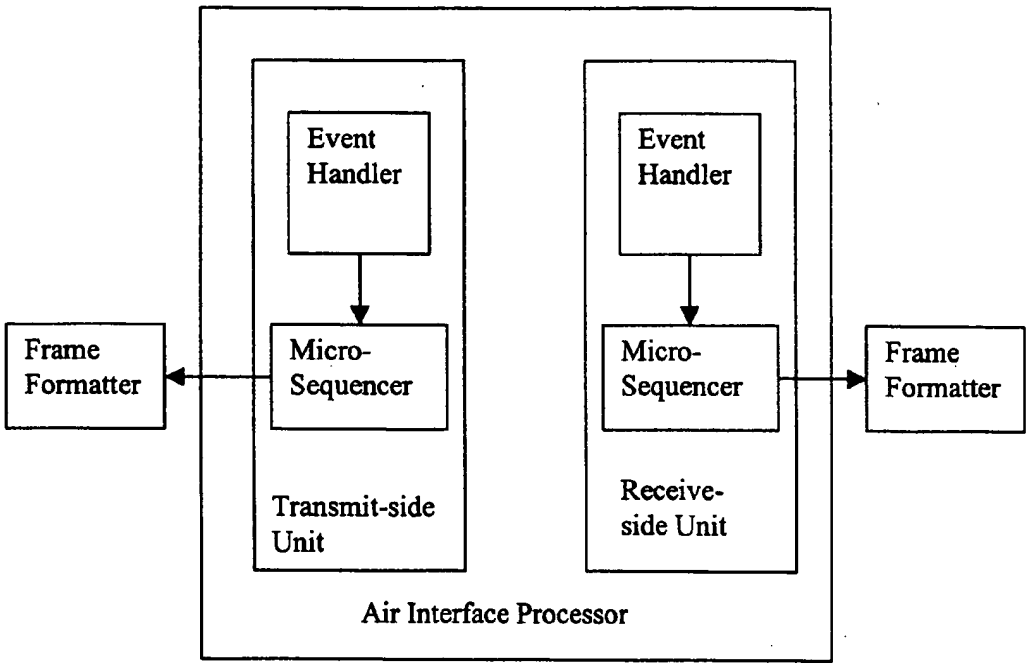
Event Handler
Event Handler
Frame Formatter
Micro-Sequencer
Micro-Sequencer
Frame Formatter
Transmit-side Unit
Receive-side Unit
Air Interface Processor

Figure 2

Instruction	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type 1	0	0	ALU opcode				S	W	Rd				Rn				operand 2										branch code								pass address												fail address			
Type 2	0	1	0	0	Rhi				-				Rlo				32-bit data																																	
Type 3	0	1	0	1	Rhi				Imm_lo								32-bit data																																	
Type 4	0	1	1	0	Rhi				-				Rlo				-																														Rd			
Type 5	0	1	1	1	Rhi				Imm_lo								-																														Rd			
Type 6	1	T	D	Q			0	Microsequencer address									trigger time																																	
Type 7	1	-			0	1	-	burst info																																										
Type 8	1	-	A	1	1	-	mask																																										

Instruction Type 1: ALU Operations
Instruction Type 2: Write register
Instruction Type 3: Write register immediate
Instruction Type 4: Read register
Instruction Type 5: Read register immediate
Instruction Type 6: Trigger
Instruction Type 7: BURST
Instruction Type 8: WAIT
[A='0' → until any of (R12 and mask) bits are set]
[A='1' → until all of (R12 and mask) bits are set]

Figure 3: Event Handler Instruction Set Summary

Immediate mode (16 bit result)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
shift_by				val											

```
shift(op2) = LSL ( val, shift_by )
```

Register mode 0 (16 bit result)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
shift_by				0	type		0	Rm							

```
shift(op2) = LSL ( Rm, shift_by ) if type=0
             LSR ( Rm, shift_by ) if type=1
             ASR ( Rm, shift_by ) if type=2
             ROR ( Rm, shift_by ) if type=3
```

Register mode 1 (16 bit result)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rs				0	type		1	Rm							

```
shift_by = Rs(3 downto 0)
shift(op2) = LSL ( Rm, shift_by ) if type=0
             LSR ( Rm, shift_by ) if type=1
             ASR ( Rm, shift_by ) if type=2
             ROR ( Rm, shift_by ) if type=3
```

Figure 4: General Purpose Instructions

Opcode		Description
0000	AND	Bitwise vector AND Rd := Rn AND op2
0001	XOR	Bitwise vector XOR Rd := Rn XOR op2
0010	SUB	Subtract Rd := Rn – op2
0011	RSB	Reverse Subtract Rd := op2 – Rn
0100	ADD	Add Rd := Rn + op2
0101	ADC	Add w/ Carry Rd := Rn + op2 + C
0110	SBC	Subtract w/ Carry Rd := Rn – op2 – NOT C
0111	RSC	Reverse Subtract w/ Carry Rd := op2 – Rn – NOT C
1000	OR	Bitwise vector OR Rd := Rn OR op2
1001	BIC	Bit Clear Rd := Rn AND NOT op2
1010	INT	Raise Interrupt
1011		reserved
1100		reserved
1101		reserved
1110		reserved
1111		reserved

Figure 5: Event Handler ALU Opcodes

0000	JEQ	equal Z
0001	JNE	not equal !Z
0010	JCS	unsigned higher or same C
0011	JCC	unsigned lower !C
0100	JMI	negative N
0101	JPL	positive or zero !N
0110	JVS	overflow V
0111	JVC	no overflow !V
1000	JHI	unsigned higher C & !Z
1001	JLS	unsigned lower or same !C | Z
1010	JGE	greater than or equal N & V | !N & !V
1011	JLT	less than N & !V | !N & V
1100	JGT	greater than !Z & (N & V + !N & !V)
1101	JLE	less than or equal Z & (N & !V + !N & V)
1110	JMP	unconditional
1111		reserved

Figure 6: Event Handler Branch Conditions

Instruction	47	46	45	44	43–40	39	38	37	36	35–32	31–4	3–0
Type 2	0	1	0	0	Rhi	-	-	-	-	Rlo	32-bit data	
Type 3	0	1	0	1	Rhi	Imm_lo					32-bit data	
Type 4	0	1	1	0	Rhi	-	-	-	-	Rlo	- -	Rd
Type 5	0	1	1	1	Rhi	Imm_lo					- -	Rd

Figure 7: Register Access Instructions

Instruction							
Type 6	1	T	D	Q	0	Microsequencer address	trigger time

Figure 8: Data Scheduling Instructions

Instruction						
Type 7	1	-	0	1	-	burst info

Figure 9: Burst Descriptor Instruction

PS	value to DDS/Fractional-N counter

Figure 10: Modulator Burst Info Field Format

User ID	PS	Expected Length

Figure 11: Demodulator Burst Info Field Format

Type 8	1	-	A	1	1	-	mask

Figure 12: Processor Wait Instruction

Opcode	Mnemonic	Description
00000	JZ	Jump to Zero
00001	CJS	Conditional Jump to Subroutine
00010	JMAP	Jump Map
00011	CJP	Conditional Jump Pipeline
00100	PUSH	Push/Conditional Load Counter
00101	JSRP	Conditional Jump to Subroutine
00110	CJV	Conditional Jump Vector
00111	JRP	Conditional Jump
01000	RFCT	Repeat Loop Counter Not Equal to Zero
01001	RPCT	Repeat Pipeline Counter Not Equal to Zero
01010	CRTN	Conditional Return
01011	CJPP	Conditional Jump Pipeline and Pop
01100	LDCT	Load Counter and Continue
01101	LOOP	Test End of Loop
01110	CONT	Continue
01111	TWB	Three Way Branch
10000	FORK	Multiway Branch
others		reserved

Figure 13: Microsequencer Instruction Set

OPCODE	EMIT	CCSEL	CP	FFCMD	SB	OC	-	SR

Figure 14: Microsequencer memory format

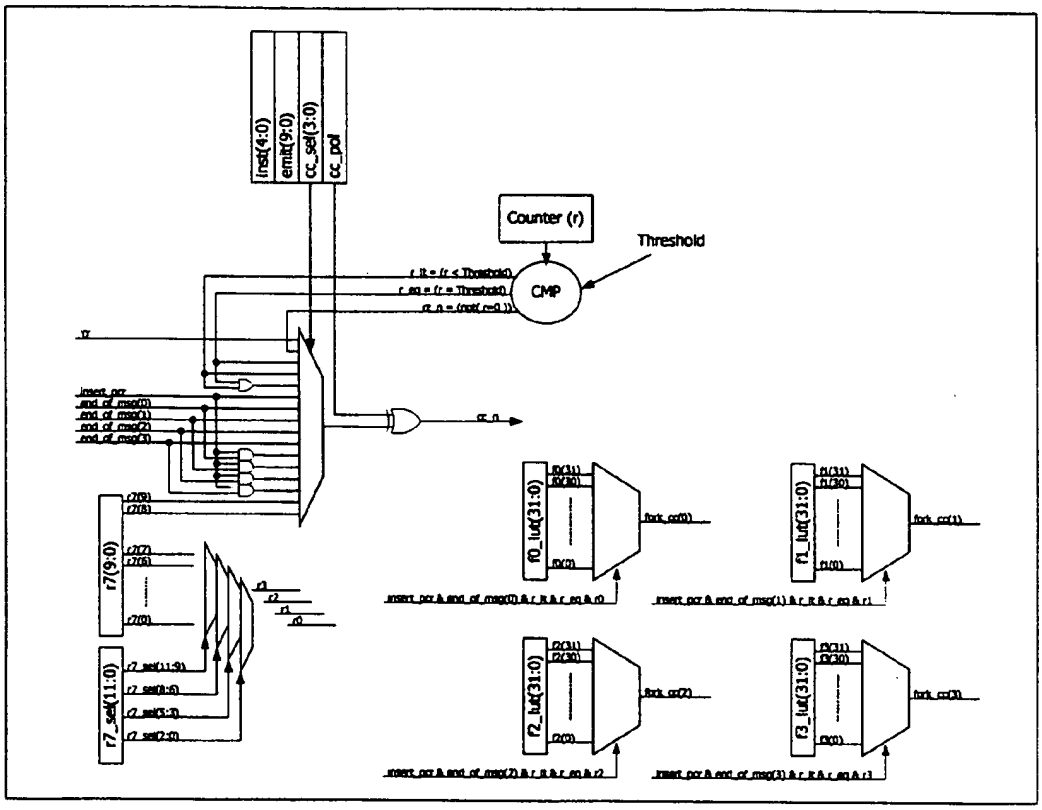

Figure 15: Configuration of condition codes and fork

0	0	0	0	0	0	0
0	0	0	0	1	0	
0	0	0	1	0	0	
0	0	0	1	1	0	
0	0	1	0	0	0	0
0	0	1	0	1	0	
0	0	1	1	0	0	
0	0	1	1	1	0	
0	1	0	0	0	0	0
0	1	0	0	1	0	
0	1	0	1	0	0	
0	1	0	1	1	0	
0	1	1	0	0	0	0
0	1	1	0	1	0	
0	1	1	1	0	0	
0	1	1	1	1	0	
1	0	0	0	0	0	C
1	0	0	0	1	0	
1	0	0	1	0	1	
1	0	0	1	1	1	
1	0	1	0	0	0	C
1	0	1	0	1	0	
1	0	1	1	0	1	
1	0	1	1	1	1	
1	1	0	0	0	0	C
1	1	0	0	1	0	
1	1	0	1	0	1	
1	1	0	1	1	1	
1	1	1	0	0	0	C
1	1	1	0	1	0	
1	1	1	1	0	1	
1	1	1	1	1	1	

Fig. 16